Scalable real-time classification of data streams with concept drift

Future Generation Computer Systems 75 (2017) 187–199

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Scalable real-time classification of data streams with concept drift

Mark Tennant [a], Frederic Stahl [a,*], Omer Rana [b], João Bártolo Gomes [c]

[a] University of Reading, Whiteknights, PO Box 225, RG6 6AY, Reading, UK
[b] Cardiff University, Computer Science & Informatics, Queen's Buildings, 5 The Parade, Roath, CF24 3AA, Cardiff, UK
[c] Institute for Infocomm Research (I2R), A*STAR, 1 Fusionopolis Way Connexis, Singapore 138632, Singapore

HIGHLIGHTS

- A real-time data stream classifier adaptive to concept drift and robust to noise.
- A parallel implementation of the real-time data stream classifier.
- A discussion about using open source Big Data technologies for data stream mining.

ARTICLE INFO

Article history:
Received 30 November 2015
Received in revised form
3 June 2016
Accepted 22 March 2017
Available online 9 April 2017

Keywords:
Parallel data stream classification
Adaptation to concept drift
High velocity data streams

ABSTRACT

Inducing adaptive predictive models in real-time from high throughput data streams is one of the most challenging areas of Big Data Analytics. The fact that data streams may contain concept drifts (changes of the pattern encoded in the stream over time) and are unbounded, imposes unique challenges in comparison with predictive data mining from batch data. Several real-time predictive data stream algorithms exist, however, most approaches are not naturally parallel and thus limited in their scalability. This paper highlights the Micro-Cluster Nearest Neighbour (MC-NN) data stream classifier. MC-NN is based on statistical summaries of the data stream and a nearest neighbour approach, which makes MC-NN naturally parallel. In its serial version MC-NN is able to handle data streams, the data does not need to reside in memory and is processed incrementally. MC-NN is also able to adapt to concept drifts. This paper provides an empirical study on the serial algorithm's speed, adaptivity and accuracy. Furthermore, this paper discusses the new parallel implementation of MC-NN, its parallel properties and provides an empirical scalability study.

1. Introduction

The 4 main aspects of Big Data are [1]: data generated at a fast rate (*Velocity*), very large and potentially unknown data quantities (*Volume*), uncertainty in the data (*Veracity*) and different forms of data such as text, structured data etc. (*Variety*). Other aspects of Big Data have been added over the years, i.e. *Volatility*, referring to how long the data is valid for, which is particularly relevant when referring to real-time data streams; and *Value*, referring to potential insights that can be derived by analysing the data. Regarding *Velocity*, data arriving at a very high speed challenges our computational hardware processing capabilities [2,3]. This paper presents an algorithm that addresses the overlap of the *Velocity* and *Volume* aspects of Big Data Analytics through a parallel and adaptive real-time data stream classifier. In data stream classification a classifier is trained in real-time on incoming labelled data instances. This classifier is then used in real-time to predict the class label of previously unseen data instances. The classifier is required to adapt to changes of concepts that can occur over time (known as concept drift [4]), in order to keep an accurate classification model over time.

The growing importance of data stream classification techniques is reflected through many commercial applications, such as: sensor networks; Internet traffic management and web log analysis [5]; TCP/IP packet monitoring [6]; intrusion detection [7]; and credit card fraud detection [8]. Due to high throughput of data and potentially infinite data streams, it is often not feasible to capture, store and process the data. In the past two decades this has led to the development and publication of data stream classifiers that can analyse the data in real-time as it is being generated. For example,

* Corresponding author.
E-mail addresses: M.Tennant@pgr.reading.ac.uk (M. Tennant), F.T.Stahl@reading.ac.uk (F. Stahl), RanaOF@cardiff.ac.uk (O. Rana), bartologjp@i2r.a-star.edu.sg (J.B. Gomes).

http://dx.doi.org/10.1016/j.future.2017.03.026

data stream classifiers such as Hoeffding Trees [9], G-eRules [10], Very Fast Decision Rules (VFDR) [11] only need one pass through the data and thus train and adapt to concept drifts in real-time scenarios. Nonetheless, their scalability is limited to the utilisation of one processing node at a time.

Few attempts have been made to combine parallelism and real-time data stream classification. Parallel binning is used by the SPDT [12] algorithm. However, the updating of SPDT classifier is not performed in parallel. Vertical Hoeffding Trees (VHDT) [13] partition the stream instances in terms of attributes, to support parallel processing. However, VHDTs scalability is limited by the number of attributes, as the attributes are distributed evenly over the number of processors utilised. In addition there exists a parallel method for concept drift detection termed Online MapReduce Drift Detection Method (OMR-DDM) [14], which makes use of the error rate of a collection of classifiers executed concurrently.

This paper proposes an inherently parallel adaptive data stream classifier termed MC-NN. The classifier is based on Nearest Neighbour (NN) classification and statistical summaries of the data and recency. The statistical summary is structured in the form of a set of variance based Micro-Clusters (MCs). Micro-Clusters continuously adapt to concept drifts through absorbing new data instances (updating statistics). An empirical evaluation [15] shows that the serial implementation of MC-NN is already very fast and robust to noise and concept drifts. However, it is limited by the throughput of a single computational node.

A parallel implementation of MC-NN is presented, along with a critical appraisal of implementation mechanisms that can be used to support parallel analysis of real-time data. A scalability evaluation is also carried out, identifying insights, difficulties and solutions in implementing parallel real-time data stream classifiers.

This paper is organised as follows: Section 2 summaries some related work. Section 3 describes the developed naturally parallel MC-NN algorithm and provides an empirical study comparing it with its serial competitors in terms of classification accuracy, adaptation to concept drifts and speed. Section 4 discusses the parallel implementation of MC-NN and provides an empirical scalability evaluation. It further discusses issues and experiences in implementing real-time data processing algorithms. Concluding remarks are provided in Section 5.

2. Related work

In the more general area of data mining an algorithm would iterate over the data several times in order to generate a model that fits the concepts (patterns) in the data. In each iteration the model is altered in order to better fit the concepts. However, as data streams are inherently infinite in length, iterative processes cannot be used. If left un-monitored, the algorithms would try to fit the concepts encoded on the whole stream and not account for *'Concept Drifts'*. 'Concepts' can be thought of as blocks of homogeneous/statistically similar data in a linear time frame. As the length and number of 'Concepts' is unknown the data stream must be monitored in real-time. An interesting area of research is the development of 'standalone' Concept Drift Detectors that monitor data streams in real-time. Typically, when a Drift Detector algorithm detects a concept drift (correctly or incorrectly) the current model is deleted and a new model is created. Examples of Concept Drift detectors are DDM [16], ECDD [17] and ELM[18]. Typically Concept Drift detectors work independently and concurrently from the underlying data mining algorithm (i.e. a classifier). Thus, both the data mining algorithm and the concept drift detector have to be computationally efficient in order not to hinder the computational performance. The data mining algorithm used in conjunction with the drift detector does not need to be adaptive — batch algorithms such as C4.5 [19], Support Vector Machines [20], N-Prism [21], KNN, etc. can also be used. This would require buffering enough data after the concept drift has occurred and then applying the batch data mining algorithm on the buffer. However, as mentioned earlier, batch algorithms typically require several passes through the data and thus may be too slow if data is arriving at a high speed. Further techniques exist to adapt non adaptive data mining algorithms to streaming data, such as sliding window [2] and reservoir sampling [22]. Reservoir sampling maintains an unbiased and representative fixed sized sample of the data instances retrieved from the stream, whereas sliding window based algorithms, such as G-eRules [10], consider only the most recent instances from the stream to build the data mining model. However, these techniques would require to re-train batch algorithms and thus may be too slow and impractical to use for data arriving at high speed.

Other techniques such as Hoeffding bound based techniques [23] and Micro-Clusters [24] have been used to create inherently adaptive data stream mining algorithms. Hoeffding based techniques aim to create and adapt data mining models based on a statistical upper bound on the probability that the so far received attribute values deviates from its expected value. The Hoeffding bound has been successfully used to create various data stream mining algorithms known as Very Fast Machine Learning (VFDT [25]). Micro-Cluster based techniques aim to create a statistical summary in terms of feature values, value distribution and time-stamps of the data retrieved from the stream (CluStream[24], On Demand Classification of Data Streams [26]).

A number of systems exist to support parallel stream processing, the most notable of these include Esper [27]. However, Esper makes use of a centralised architecture that runs on a single node and keeps everything (states, operators, and so on) in memory (although support is provided for multi-threading). However, if the continuous queries have a large window size and might require processing of a large number of data items/sec, Samza [28] provides a better alternative [29]. Samza can be though of as a simplified 'pilot job' [30]. In the pilot terminology a Samza job is an 'early bound' container that will process an unknown future workload. The key difference is that a pilot task is identified as an individual with its own workload to process, and therefore has no interaction with other tasks. Samza containers bound to a data stream are static, they have access to all data passed through the underlying stream until they are stopped externally. Other alternatives with similar functionality include: Apache Storm, Spark Streaming and Apache S4 — a comparison can be found in [29]. A number of messaging systems exist that Samza is capable of utilising. Broadly speaking they can be split into three groups: *Message Queue Systems*, such as Kestrel and RabbitMQ [31], *Publish Subscribe Systems*, such as Kafka [32] and Kestrel [33], and *Log Systems*, such as Flume [34] and Scribe [35]. The work presented in this paper develops an inherently parallel and adaptive data stream classifier that makes use of parallel stream processing technologies.

3. Adaptive Micro-Cluster nearest neighbour data stream classification

3.1. Micro-Cluster based nearest neighbour

In the authors' previous feasibility study [36], a real-time classifier was implemented based upon KNN. In KNN a data instance is assigned the class that is most common amongst its K Nearest Neighbours. The basic approach of the real-time KNN is to keep a sliding fixed sized time window of the most recent data instances and execute KNN from the sliding window set. Real-time KNN retrains on recent instances whilst older instances are deleted. However, real-time KNN is computationally limited

by faster data streams [36]. To overcome the computational bottleneck of real-time KNN and the problems associated with the sliding window, the presented classifier adapts Micro-Clusters [37] — a technique originally developed for data stream clustering [24] in order to provide a summary of the locality of the data are of the form:

$\langle CF2^x, CF1^x, CF2^t, CF1^t, n\rangle$

The notation used for Micro-Cluster has been taken from [24], the original paper that introduced Micro-Clusters. The sum of the squares of the attributes are maintained in vector $CF2^x$, the sum of the values in vector $CF1^x$; the sum of time stamps in vector $CF1^t$; and the number of data instances is stored in scalar n. In this notation CF stands for Cluster Feature, the superscripts x and t denote if the CF is storing statistics about feature values or their time stamps respectively. The numbers 1 and 2 used in the CF notation denote if the CF stores the sum of the feature values or the squared sum respectively. $CF2^x$ and $CF1^x$ can be used to calculate the locality and boundary of the Micro-Clusters whereas $CF2^t$ and $CF1^t$ can be used to determine the *recency* of the data summarised in the cluster. MC-NN adapts Micro-Clusters to compute nearest neighbours for classification. The Micro-Cluster structure has been extended by terms CL for the cluster's class label, ϵ as error count, Θ as error threshold for *splitting*, α as initial time stamp and Ω as a threshold for the Micro-Cluster's performance — leading to the extended notation:

$\langle CF2^x, CF1^x, CF1^t, n, CL, \epsilon, \Theta, \alpha, \Omega\rangle$

The centroid of the Micro-Cluster can be calculated by $\frac{CF1^x}{n}$. In order to classify a new data instance from the stream the MC-NN classifier calculates the Euclidean distances between the data instance and each Micro-Cluster centroid and the class label of the nearest Micro-Cluster is assigned to the data instance. In our experiment, Euclidean distance has been demonstrated to work faster than alternative distance measures, while achieving a competitive accuracy. ϵ of a Micro-Cluster is initially 0 and incremented by 1 if the Micro-Cluster is used for classification and miss-classifies the data instance. Likewise ϵ is decremented by 1 if the Micro-Cluster is involved in a correct classification. Θ is a user defined upper limit of acceptable ϵ. It is expected that a low Θ will cause the algorithm to adapt to changes faster, but will be more susceptible to noise. A larger Θ value will be more tolerant to noise but may not 'learn' as fast.

As more labelled instances are received for learning they will change the distribution of the Micro-Clusters. According to Algorithm 1 two scenarios are possible after the nearest Micro-Cluster has been identified when a new training instance is presented to the classifier:

Scenario 1: If the nearest Micro-Cluster is of the same label as the training instance, then the instance is incrementally added to the Micro-Cluster and ϵ is decremented by 1.

Scenario 2: If the nearest Micro-Cluster is of a different class label, then the training instance is incrementally added to the nearest Micro-Cluster that matches the training instance's class label. However, the error count ϵ of both involved Micro-Clusters is incremented.

If over time a Micro-Cluster's error count ϵ reaches the error threshold Θ, then the Micro-Cluster is *split*. This is done by evaluating the Micro-Cluster's dimensions for the size of its variance, which can be calculated using Eq. (1), where x denotes a particular attribute. The splitting of a Micro-Cluster generates two new Micro-Clusters, centred about the point of the parent Micro-Cluster's attribute of greatest variance; while the parent Micro-Cluster is removed. The two new Micro-Clusters inherit all configuration values from the parent. Therefore, all future classifications made by these new Micro-Cluster's (or their children) will be of the same class label. The assumption behind this way of splitting attributes is that a larger variance value of one attribute over another indicates that a greater range of values have been seen for this attribute. Therefore, the attribute may contribute towards miss-classifications. This splitting of a Micro-Cluster causes the two new Micro-Clusters to separate and better fit the underlying concept encoded in the stream. Once the attribute of largest variance has been identified, the two new Micro-Clusters are initially populated with the parent's internal mean/centre data ($CF1^x$). The split attribute (with the largest variance), is altered by the variance value identified in the positive direction in one of the new Micro-Clusters and negatively in the other. This ensures that future training will further re-position the two new Micro-Clusters better than the parent could alone.

Data: Train Instance
Result: Re-Positioned Localised sub-set of Micro-Clusters
Remove Micro-Clusters with poor performance (under Ω value)
foreach *Micro-Cluster in LocalSet* **do**
  Evaluate Micro-Cluster against NewInstance;
end
Sort EvaluationsByDistance();
if *Nearest Micro-Cluster is of the Training Items Class Label* **then**
  CorrectClassification Event
  NewInstance is Incremented into Nearest Micro-Cluster
  Nearset Micro-Cluster Error count (ϵ) reduced.
else
  MisClassification Event
  2 Micro-Clusters Identified:
  1) Micro-Cluster that should have been identified as the Nearest to the New Instance of the same Classification Label.
  2) Micro-Cluster that incorrectly was Nearest the New Instance.
  Training Item incrementally added to Micro-Cluster of Correct Classification Label. Both Micro-Clusters have internal Error count (ϵ) Incremented
  foreach *Micro-Cluster Identified* **do**
    if *Micro-Cluster Error count (ϵ) exceeds Error Threshold (θ)* **then**
      Sub-Divide Micro-Cluster upon attribute of largest Variance
    end
  end
end

Algorithm 1: Training the MC-NN classifier

$$Variance[x] = \sqrt{\left(\frac{CF2^x}{n}\right) - \left(\frac{CF1^x}{n}\right)^2} \quad (1)$$

When a Micro-Cluster has a new instance added to it, its internal instance count n is incremented by 1 and the sum of time stamps ($CF1^t$) is incremented by the new time stamp value(T). The *Triangular Number* $\Delta(T)$ (Eq. (2)) of this time stamp will give an upper bound to the maximum possible value of $CF1^t$. Therefore, if all instances were entered into this Micro-Cluster $CF1^t$ would be equal to the triangular number of T. The lower the value of $CF1^t$ is from the *Triangular Number* the poorer the Micro-Cluster has been participating in the stream classification. The use of Triangular Numbers gives more importance to recent instances over earlier ones added to the Micro-Cluster, as the time stamp value (T) is always increasing and MC-NN uses the sum of these incremental values. Triangular numbers assume that all Micro-Clusters were created at time stamp 1. To counter this, each Micro-Cluster keeps track of the time stamp when it was initialised (α). The Micro-

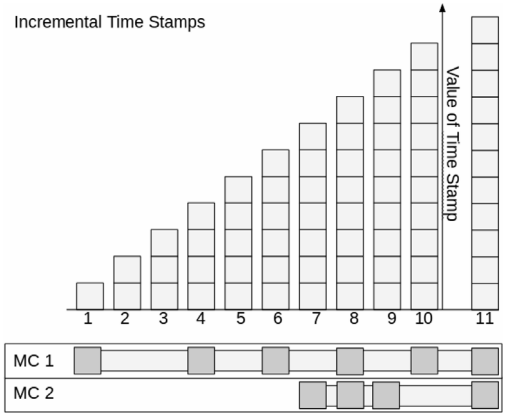


Fig. 1. Incremental time stamps and Triangular Number propagation. In this example 11 is the most recent time stamp and 1 the first time stamp recorded. The grey boxes for MC1 and MC2 denote that at this particular time stamp a data instance has been absorbed by the Micro-Cluster.

Cluster's real $\Delta(T)$, denoted $\Delta(T)_{real}$, can be calculated by $\Delta(T) - \Delta(\alpha)$. Any Micro-Clusters that fall under a pre-set threshold value of (Ω) are deleted as they are considered old. For the rest of this paper a value of 50% was given to all Micro-Cluster Ω values as it seemed to work best for most classification problems.

$$\textit{Triangular Number } \Delta(T) = ((T^2 + T)/2) \tag{2}$$

At any point in the data stream the Triangular Number (Eq. (2)) can be calculated for each Micro-Cluster.

For example, consider the two Micro-Clusters (MC1 and MC2) as depicted in Fig. 1. MC1 was created at time stamp 1 and was updated with instances at time stamps 4, 6, 8 and 10. MC2 was created at time stamp 7 and updated with instances at time stamps 8 and 9. This example evaluates the Micro-Clusters' participation at time stamp 11.

Calculate the Triangular number for Time Stamp 11.

$$\textit{Triangular Number } \Delta(11) = ((11^2 + 11)/2) = 66$$

Calculate the initial Triangular number for each Micro-Cluster:

$$MC1\ \Delta(1) = ((1^2 + 1)/2) = 1$$
$$MC2\ \Delta(7) = ((7^2 + 7)/2) = 28$$

Calculate each Micro-Cluster's real Triangular Number:

$$MC1\ \Delta_{real} : (66 - 1) = 65$$
$$MC2\ \Delta_{real} : (66 - 28) = 38$$

Retrieve each Micro-Cluster's actual time stamp values:

$$MC1 : (1 + 4 + 6 + 8 + 10 + 11) = 40$$
$$MC2 : (7 + 8 + 9 + 11) = 35$$

Calculate each Micro-Cluster's participation percentage:

$$MC1 : (40 * 100/65) = 61\%$$
$$MC2 : (35 * 100/38) = 92\%$$

If either of these percentages drops below the performance threshold (Ω) the Micro-Cluster is deleted. It is interesting to note that the use of the Triangular Number naturally biases towards later instances in the data stream. In the example, MC1 contained 5 instances whereas MC2 only contained 3. Due to the fact that MC2's instances happened later in the data stream gives it 92% participation score against MC1's 61% participation, even though it has a 50% (5 out of 10) actual instance insertions. This means that as the data stream progresses, a Micro-Cluster's participation percentage can increase; as past 'gaps' become less relevant to the overall scoring, which is desirable in adaptive classifiers.

3.2. Evaluation of MC-NN's adaptation to concept drifts and classification accuracy

3.2.1. Complexity of MC-NN

The complexity definitions used in this section are given in Table 1. When implementing the MC-NN algorithm the number of Micro-Clusters per class label is fixed at the maximum (mx). In the experiments discussed in this paper mx was set to 25 Micro-Clusters for each class label. Once the Micro-Clusters are initialised, the number of Micro-Clusters never changes and thus the memory footprint remains constant. The algorithm begins with one Micro-Cluster per class label and utilises additional ones from the existing pool of Micro-Clusters depending on the data stream. This is done in order to facilitate efficiency at runtime. The creation and deletion of Micro-Clusters is implemented using array pointers and counters. The memory size is therefore always a constant: $O(mx * c * d)$.

During training the distances of new instances are calculated against each of the Micro-Clusters' centroids, and the nearest Micro-Cluster' class label is used for classification. This is equivalent to computing the Euclidean distance between 2 vectors, where each vector value corresponds to a data attribute and there are d attributes (dimensions). This results in a complexity dependent upon both the number of Micro-Clusters per class label and the dimension of the data: $O(m * c * d)$.

In the best case, where there is only one Micro-Cluster per class label this will be $\Omega(ml)$ or $\Omega(c)$, in the worst case this will be limited to the number of Micro-Clusters allowed by the algorithm $\Theta(mx * c)$. For example Fig. 2(a) displays a linear distribution of data ($c = 2, d = 2, n = 20$). Fig. 2(b) displays how each of the data points (n) will be inserted into each Micro-Cluster (m) to determine their centroids. This shows the best case complexity for new instances to be tested against $\Omega(c)$, where $c \ll n$. For a non-linear data pattern as shown in Fig. 2(c) ($c = 2, d = 2, n = 20$), the data is unable to be classified correctly with just 2 Micro-Clusters. Fig. 2(d) illustrates that multiple Micro-Clusters from each class label will be required to fit the data. This shows that the complexity of this data pattern is $O(m)$, where again $m \ll n$.

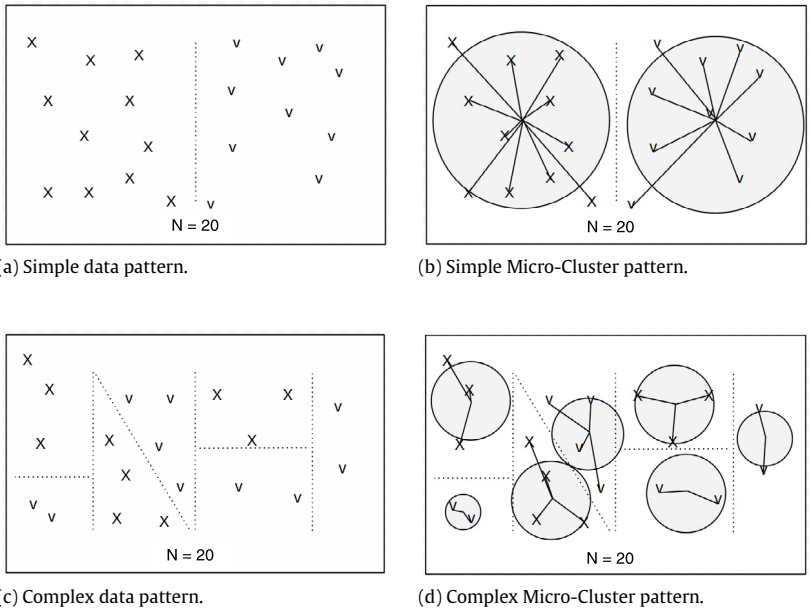
3.2.2. Experimental setup

For the evaluation of this serial MC-NN implementation, an 'Intel core' I5 processor with 8 Gb RAM was used. All synthetic data generators and algorithms evaluated in Section 3.2 were taken from the Massive Online Analysis (MOA) framework [38]. Also the here presented real-time KNN and MC-NN algorithms were implemented within the MOA framework.

Four data streams have been utilised: The **SEA data stream** [39] contains three continuous attributes and two class labels. The stream can be set to 1 of 4 function parameters. The function selected determines the value of a sum threshold of the first 2 continuous attribute values (threshold values: 8,9,7,9.5). A class label of *True* is given only if the threshold level is surpassed, otherwise class label *False* is given (i.e. $attribute1 + attribute2 < threshold$). Arbitrarily function 1 was chosen for the initial concept and function 3 for the concept change. The **Random Tree Generator** [25] creates a random tree with split points using the attributes associated with the stream. Each tree leaf is labelled with class labels and then all subsequent randomly generated instances are labelled according to their traversal of the tree to a leaf node. In our experiments the random tree(s) comprise ten continuous attributes and three distinct class labels. A drift is achieved by simply generating a different random tree. Both the Random Tree and the SEA datastreams generated 35,000 instances. The concept drift begins at instance 10,000 with a gradual change, where both concepts are present over a period of 1000 instances. Random Trees generate useful data streams as the number of

Table 1
Complexity definitions.

n	The number of instances in the data stream (only limited by experimentation as infinite in real data stream)
d	The number of dimensions of the data (number of attributes in each instance)
c	Then number of class labels to classify
mx	The maximum number of Micro-Clusters (default set to 25 per class label)
m	The number of Micro-Clusters required to cover a specific data pattern
ml	The minimum number of Micro-Clusters (one Micro-Cluster per class label)



(a) Simple data pattern.
(b) Simple Micro-Cluster pattern.
(c) Complex data pattern.
(d) Complex Micro-Cluster pattern.

Fig. 2. Complexity of Micro-Clusters.

attributes and class labels are user defined, and the complexity (or level) of the tree can be altered to create a more dense data pattern. This allows for multilevel stress testing of algorithms, as denser trees usually create harder problems for classification. We expect the Hoeffding Tree to perform exceptionally well against the Random Tree data streams, as the Hoeffding Tree algorithm utilises the same underlying structure for data classification. This was a major reason for choosing this data stream as a benchmark against MC-NN. The **Hyperplane generator** creates a linearly separable model. A Hyperplane in 'D' dimensions slowly rotates continuously changing the linear decision boundary of the stream. This constant concept change makes it very difficult for data stream classifiers to keep a good classification accuracy and computational efficiency. The experiments using the Hyperplane generator created 10 million data instances, with five numerical attributes and two classes. In order to add an additional challenge 10% noise was generated as well as with probability $P(0.75)$ chance of reversing the direction of the rotation causing an 'Oscillation' effect. To contrast, a version of the stream with probability $P(0)$ chance of reversing the direction of the concept drift was also created. The **Human Activity Recognition** data set **(HAR)** [40] containing data from multiple users, performing multiple tasks, using personal smart phones/watches was used as a more realistic alternative to the generated data set to show how the algorithm could be applied in a different context. The data from both the accelerometers and gyroscopes within such personal devices is logged in microseconds whilst users are performing an array of tasks. These tasks (class labels) include: *Biking*, *Sitting*, *Standing*, *Stairs_up*, *Stairs_down*, *Walking and null*. A sub-set of this data was taken (one single users data) and implemented as a data stream containing 1.3 million instances. Each Instance has the attributes of x, y, z accelerometer data, and Device Model Identifier.

MC-NN was compared against two state-of-the-art data stream classifiers, Hoeffding Trees [9] and incremental Naïve Bayes and against its predecessor real-time KNN classifier [36]. Each instance was tested upon the classifier to log the classifier's performance before being used for training: this is also know as Prequential testing [41]. Prequential testing deviates from traditional data mining predictive accuracy testing, as stream data cannot be divided beforehand into the classic groups of 'Train' and 'Test'. Instead each training instance is used first for training and then as validation data to ascertain the predictive accuracy of the model prior to being incorporated into the model itself. A study of 'Prequential' data stream classification with drift detectors can be found in [42].

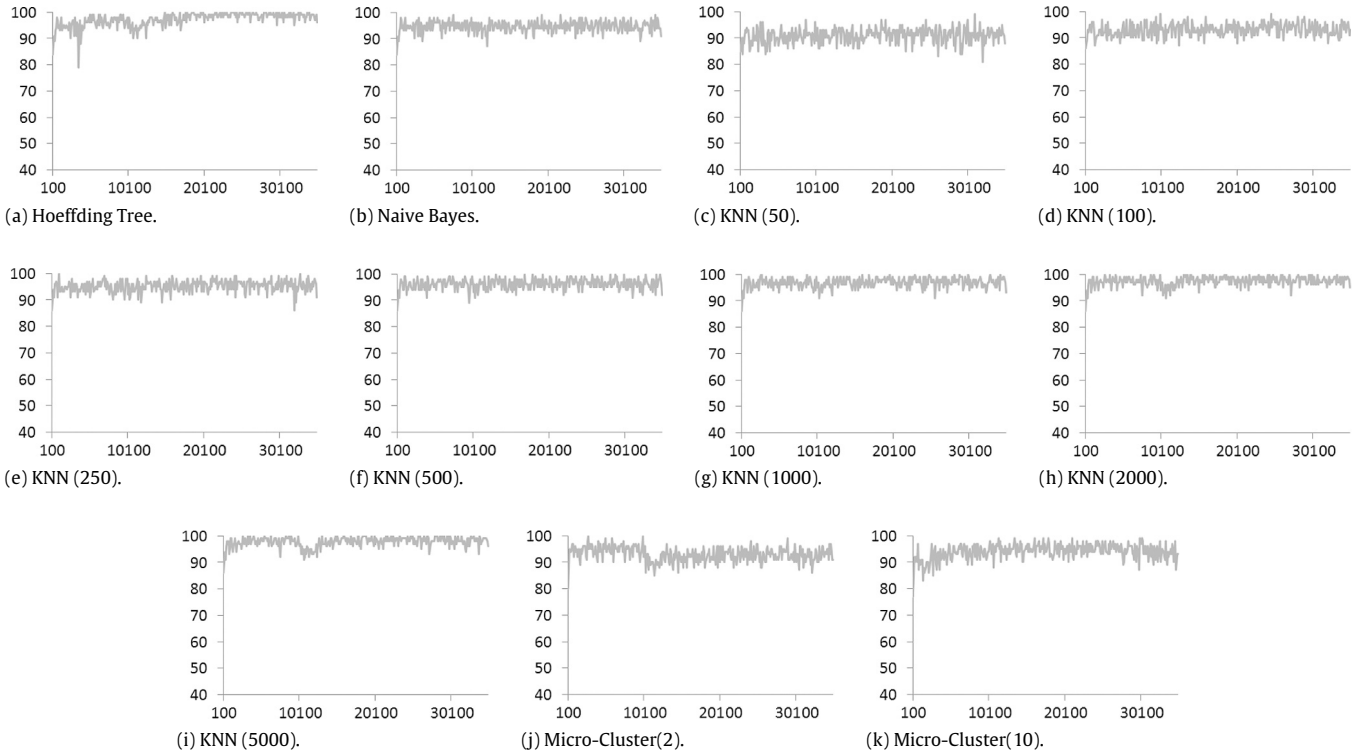
Naïve Bayes and Hoffeding Tree classifiers were chosen as they are widely covered in the literature and often considered as the best all round classifiers in data stream mining, providing a benchmark for comparison and evaluation. Although dated, both are still widely used, as they often deliver an exceptional classification performance in many applications. The Multinomial Naïve Bayes version was used recently as an ensemble to win the 2014 data mining Kaggle competition [43]. Hoffeding Trees (VFDT) were recently adapted in [44], to be used in a parallel forest executed within a GPU-based system.

3.2.3. Results for serial MC-NN implementation

Adaptation to new concepts: Two MC-NN classifiers were created, one with $\Theta = 2$ (error threshold) and the other with $\Theta = 10$. Table 2 shows how the MC-NN algorithm performs with respect to other stream classification algorithms on the SEA and Random Tree data streams. The results show that real-time KNN's results are competitive to the Hoeffding Tree and Naïve Bayes classifiers for larger K values only. However, real-time KNN is

Table 2
Accuracies and runtime of MC-NN compared with other data stream classifiers. Accuracies are listed in percent and runtime is listed in seconds. K denotes the number of nearest neighbours used in KNN and Θ the error threshold used in MC-NN.

Algorithm	SEA accuracy (runtime)	Random Tree accuracy (runtime)
Naïve Bayes	94.40(0.11)	64.17(0.10)
Hoeffding Tree	95.96(0.19)	69.88(0.28)
Real-time KNN($K = 50$)	90.46(0.27)	63.40(0.16)
Real-time KNN($K = 100$)	92.61(0.39)	65.12(0.34)
Real-time KNN($K = 250$)	94.77(1.08)	67.34(0.94)
Real-time KNN($K = 500$)	95.86(2.26)	68.84(2.05)
Real-time KNN($K = 1000$)	96.43(4.82)	70.49(4.46)
Real-time KNN($K = 2000$)	96.92(10.00)	**71.34**(**9.04**)
Real-time KNN($K = 5000$)	**97.17**(**24.73**)	65.03(24.46)
MC($\Theta = 2$)	94.03(0.28)	70.30(2.02)
MC($\Theta = 10$)	92.99(0.03)	60.99(1.49)



(a) Hoeffding Tree. (b) Naive Bayes. (c) KNN (50). (d) KNN (100).

(e) KNN (250). (f) KNN (500). (g) KNN (1000). (h) KNN (2000).

(i) KNN (5000). (j) Micro-Cluster(2). (k) Micro-Cluster(10).

Fig. 3. Concept drift adaptation on the SEA data stream. Accuracy is plotted along the vertical axis, instance stream is plotted along the horizontal axis.

multiple times slower than the well established Hoeffding Tree and Naïve Bayes approaches. MC-NN achieves accuracies close to all competitors, while clearly outperforming real-time KNN in terms of runtime. MC-NN provides a similar accuracy compared with Hoeffding Trees and Naïve Bayes, however, unlike its competitors is naturally parallel and thus can be scaled up to high speed data streams as will be shown in Section 4. It is also noticeable that a larger Θ results in a shorter runtime of MC-NN. This can be explained by the fact that when Θ is larger it will take more time for a Micro-Cluster to reach Θ and thus it will perform splits less frequently.

Figs. 3 and 4 illustrate the same experiments as listed in Table 2, however, the accuracy is displayed over time. For the SEA results displayed in Fig. 3 it can be seen that all classifiers achieve a relatively high accuracy at any time and only show a slight deterioration in accuracy during the concept drift (instances 10,000–11,000). For the Random Tree results displayed in Fig. 4 it can be seen that Hoeffding Tree and Naïve Bayes classifiers are clearly challenged with adapting to the concept drift (again between instances 10,000–11,000) as they need a long time to fully regain their previous classification accuracy level. The real-time KNN classifiers also have a noticeable deterioration of their classification accuracy during the concept drift but recover much faster compared with Hoeffding Tree and Naive Bayes. However, they do not reach the same level of classification accuracy as Hoeffding Trees and Naive Bayes. The results of MC-NN clearly show the lowest classification accuracy deterioration and almost recover instantly. MC-NN is able to reach the same classification accuracy levels as Hoeffding Tree and Naive Bayes, whereas real-time KNN performs poorly. Given the fact that MC-NN is also fast and naturally parallelisable makes MC-NN the best performing classifier on the Random Tree data stream.

Adaptation to continuous concept drift: The results in Table 3 show the total accuracy of the different classifiers evaluated on the Hyperplane data streams. The Hyperplane is particularly challenging as the concept is constantly changing and additional noise (10%) has been added. One version adds the probability of an oscillation effect with $P(0.75)$. In the other version there is no oscillation effect with P(0). In terms of classification accuracy it can be seen that MC-NN(10) achieves second highest accuracy, but is only 0.04% behind Naïve Bayes on the stream with no

Table 3
Accuracy and runtime on the Hyperplane data streams.

Algorithm	Oscillating Hyperplane % (s)	Rotating Hyperplane % (s)
Naïve Bayes	75.07(10.17)	89.89(10.12)
Hoeffding Tree	78.27(27.1)	87.43(29.44)
Real-time KNN($K = 250$)	78.54(332.0)	78.21(311.0)
MC-NN(2)	74.59(11.61)	72.0(11.7)
MC-NN(10)	87.48(9.91)	89.85(9.72)

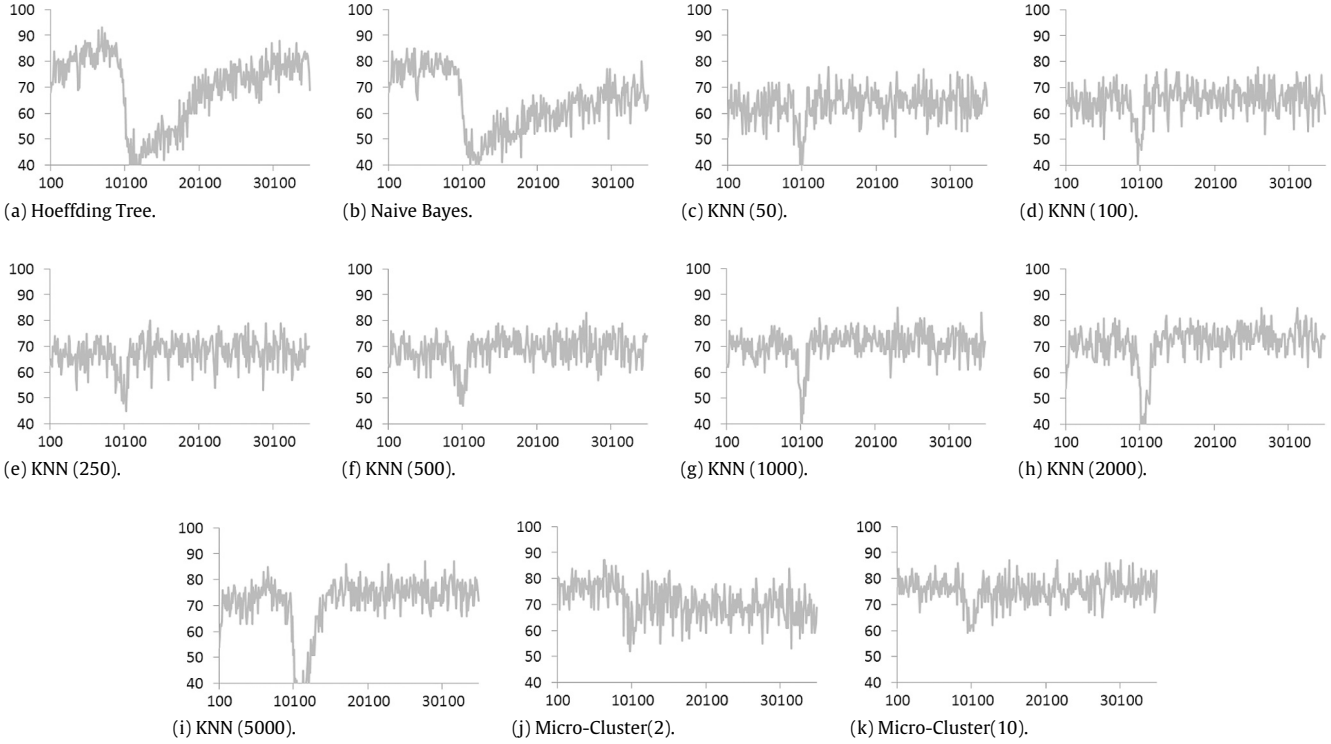


Fig. 4. Concept drift adaptation on the Random Tree data stream. Accuracy is plotted along the vertical axis, instance stream is plotted along the horizontal axis.

oscillation. On the stream with oscillation effect MC-NN(10) clearly outperforms all its competitors. Note that Table 3 displays only the runtime for the best configurations with real-time KNN. In terms of runtime, MC-NN is faster than Hoeffding Trees and achieves a similar speed to that of Naive Bayes. However, MC-NN is clearly faster (approximately 30 times faster) than real-time KNN. For the larger Θ, MC-NN performs slightly faster, which can be explained by MC-NN being less likely to perform Micro-Cluster splits which consume some of the runtime.

Fig. 5 shows the accuracy associated with experiments displayed in Table 3 for the Rotating Hyperplane data stream over time for all 10 million data instances. All classifiers need some initialisation phase before producing a stable classification accuracy. This initialisation phase is relatively short for all classifiers except MC-NN with a low Θ. Overall MC-NN(10) achieves similar performance to Naive Bayes and outperforms its predecessor real-time KNN.

Fig. 6 shows the accuracy associated with experiments displayed in Table 3 for the Oscillating Hyperplane data stream over time for all 10 million data instances. On this more challenging data stream, MC-NN(10) is stable and clearly outperforms all its competitors. Both Naïve Bayes and the Hoffeding Tree classifiers suffer at the beginning of the data stream with a negative accuracy trend. This is due to the overlapping data values that are contradicting each other due to oscillation and thus the classifiers' inability to identify a fixed separation of classes without growing an overly large and complex model.

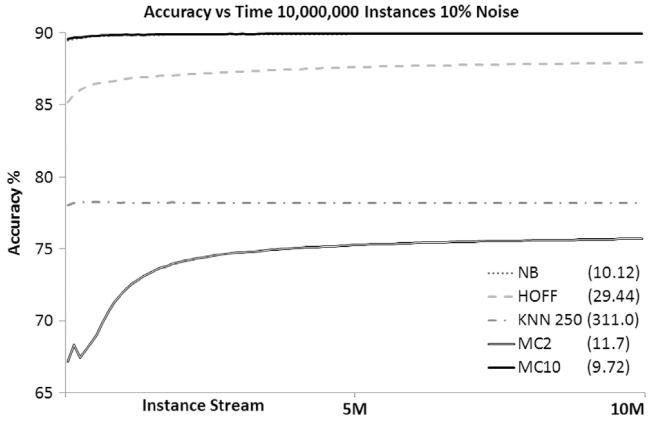


Fig. 5. Concept drift adaptation on the Hyperplane with rotating boundary on a stream length of 10 million instances.

For example, decision trees will adapt to a constantly changing concept by growing more subtrees from leaf nodes. As the Hyperplane streams are constantly changing, leaf nodes will receive conflicting data and thus the tree grows constantly. This leads to increased complexity and a high computational cost. A tree using the oscillating hyperplane will cause the tree to grow in multiple directions; whereas the rotating hyperplane will cause the tree to grow constantly one sided in the direction of the drift. Contrary to decision trees, MC-NN aims to retain its Micro-Clusters

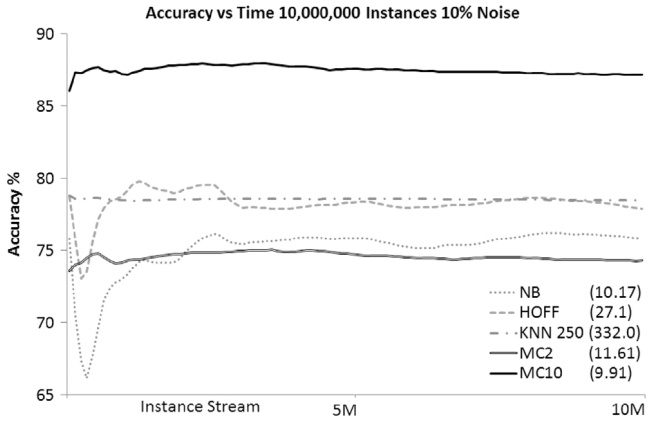


Fig. 6. Concept drift adaptation on the Hyperplane with oscillating boundary on a stream length of 10 million instances.

Table 4
Accuracy and runtime on the HAR data streams.

Algorithm	Accuracy (runtime)
Naïve Bayes	65(4.55)
Hoffeding	79(**4.38**)
MC($\Theta = 2$)	**94**(27.31)
MC($\Theta = 10$)	84.52(25.83)

by updating their statistical properties. Thus, continuous concept drifts naturally get absorbed by the model without additional computational cost for growing the model. Data streams with only occasional drifts, such as the ones illustrated in Figs. 3 and 4 will not pose this challenge for data stream classifiers. Hence, MC-NN shows a better performance on these more challenging constantly drifting data streams.

Table 4 summarises experiments of the use of MC-NN on the HAR data set. In the results it can be seen that MC-NN outperforms its competitors in terms of accuracy. However, it needs more time to process the data stream with a larger number of class labels (7), compared with the experiments on the SEA and Random Tree data streams. In the complexity analysis in Section 3.2.1 it was stated that the number of Micro-Clusters required is partially dependent on the number of class labels. Hence, processing the HAR data set requires more Micro-Clusters compared with the other data streams used in this research. However, as will be shown in Section 4.5 MC-NN can be parallelised and thus the runtime reduced.

Overall the experiments in this section showed that MC-NN clearly outperforms its predecessor in terms of classification accuracy and computational efficiency. MC-NN achieves a similar performance compared with well established data stream classifiers in terms of accuracy and runtime. However, it is more robust in terms of adaptation to concept drifts, especially complex continuous concept drifts. Moreover MC-NN is naturally parallel and thus has the advantage to be scaled up to high speed data streams as will be discussed on greater detail in Section 4.

4. Parallel MC-NN for scalability to fast data streams

This section presents a parallel implementation of the MC-NN classifier highlighted and evaluated in Section 3. The underlying idea behind the parallel implementation is that each node in a computer cluster is training (updating) Micro-Clusters individually; and each computer node is computing its similarity to newly arrived labelled data instances.

Parallel MC-NN is implemented using the *MapReduce* parallel programming paradigm, which divides computational tasks into smaller subtasks implemented as *Mappers*. Mappers are then distributed and executed concurrently on a computer cluster. The results of the Mappers are aggregated by *Reducer* components, which again can be executed concurrently in the computer cluster.

In the parallel MC-NN implementation each Mapper in the cluster computes its own MC-NN cluster set. Predictions from individual Mappers are aggregated in Reducer nodes, which assign the final class label based on the majority vote. This paper is primarily concerned with the **scalability** of data in terms of Volume and Velocity, to this end the parallelism is not simply limited to the algorithm. The data stream generation, in addition to the classification methodology has been scaled-up. To accomplish this, multiple stream generators were introduced into the system to increase the velocity of data being processed in parallel. The authors believe this to be the norm of current and future data streams (e.g. tweets, status updates and automated sensor data).

The parallel MC-NN implementation can be described in three steps, the Micro-Cluster initialisation, the training/adaptation of Micro-Clusters and the prediction of newly arrived unlabelled data instances (testing). Sections 4.1–4.3 highlight these three steps.

4.1. Initialisation

MC-NN requires a set of parameters for Micro-Clusters to adapt to data patterns and data stream variations, as discussed on Section 3.1. These configuration values are the predefined values for error count (ϵ), error threshold (θ), time stamp threshold (Ω), data stream parameters such as attribute numbers, class labels and output stream parameters (identification of Stream Generators), for returning data classifications. In the initialisation phase, these parameters are communicated to the individual Mappers, as depicted in Fig. 7. Initially each Mapper has a single Micro-Cluster for each class label, however, more clusters are likely to be generated during training and adaptation on new training instances.

4.2. Training and adaptation

Training of individual nodes only requires a single 'send' operation and no 'round trip' response time for returning predictions. Utilising a computer cluster incurs communication and management overheads. The impact of these overheads can be reduced by batching data instances together in single messages when sending them to the computer cluster. The instances in the batch are then distributed evenly inside the computer cluster across the individual Mappers (Fig. 8).

Once a labelled training instance arrives at a Mapper, the Mapper's Micro-Clusters absorb the new data instance as it is described for serial MC-NN in Section 3.1 and subsequently may split into more Micro-Clusters or delete older less participating clusters in order to adapt to concept drifts.

4.3. Prediction

Testing of a data instance requires each Mapper to be sent a copy of the unlabelled instance. A broadcast message is created containing the instance to be tested. This is depicted in Fig. 9, where the message with the instance is denoted with the letter T. Memory is allocated at the point of creation for the response of the predicted classification label from the cluster. T is broadcast to all Mappers. Upon arrival, each mapper predicts the class label of the data instance using its local configuration of Micro-Clusters as described in Section 3.1. Each of the Mappers forwards its predicted classification label to the Reducer (denoted $V1, V2, \ldots, Vn$ in Fig. 9). The Reducer node accumulates all the predictions for T and selects the classification label with the majority prediction. The classification label result (denoted R in Fig. 9) is sent back to the originator of T.

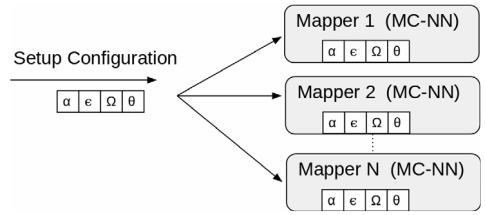


Fig. 7. Parallel initialisation setup.

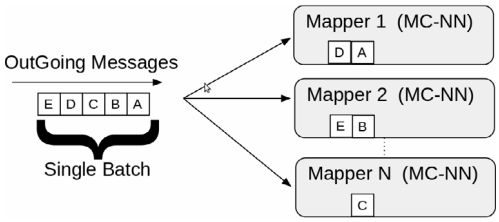


Fig. 8. Batch training message distribution.

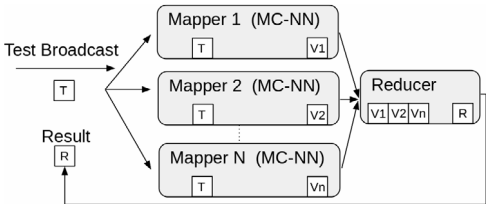


Fig. 9. Test instance cycle.

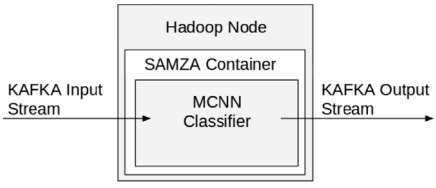


Fig. 10. Architecture of a MC-NN computer cluster node.

4.4. Computer cluster architecture for parallel scalable MC-NN

The architecture utilises multiple open source technologies to handle real-time data stream processing. The cluster consists of 17 rack mounted physical servers. Each node consists of a quad core machine running CentOS 5. Fig. 10 illustrates the general architecture of a typical MC-NN computer cluster node.

The architecture is centred on the use of Hadoop, which was initially designed to work with files and batch processing on large parallel processing tasks (e.g. distributed file searching and word count problems), the later versions of Hadoop (2.0 — YARN) can accept data from different sources such as MPI and Kafka [32]. Hadoop is designed to process specific processing jobs with recursive/interactive access to all the data available (usually within the distributed file system). The cluster used for this evaluation hosted Hadoop version 2.6.0.

Samza [28], is designed to enable Data Stream Processing over the Hadoop framework. Samza creates and launches processing jobs of unknown continuous duration within the Hadoop framework. Instead of a traditional Hadoop job running until all data has been processed in synchronised Map/Reduce stages the Samza job awaits for data to arrive from an external message system. Spark had been considered as alternative to Samza for this research. Samza was chosen as the stream processing engine because it has been designed with stream processing in mind, while Spark achieves stream processing through time based windowing of data objects.

The message system used in the experiment was **Kafka** [32], as publish subscribe systems can be manipulated more accurately for multiple stream configurations; such as different Hadoop configurations. Kafka has low latency/high throughput message handling and is also configurable to have its internal data streams (known in Kafka as 'Topics') individually distributed in parallel as required. Kafka can handle very long and fast streams due to fast communication and distribution by employing a number of low latency techniques. Such techniques are for example: retaining a maximum number of messages (containing stream data instances) in memory, and using pointers for tracking data stream consumers (i.e. MC-NN's Mappers) to speed up the reading of data. The original messages are retained within memory, while a secondary process thread is tasked with saving the data to hard disk. Multiple configurations of data producers and consumers can be created to best maximise the type of data throughput required by individual tasks. Multiple messages can be batched together to reduce the communications overhead. In the following experiments in Section 4.5, 10,000 data instances were batched together in a message to reduce communications latency to experimentally 'stress' the cluster. This batching is to highlight the performance of the algorithm, not the limitations of the hardware setup. As Kafka is a distributed system in itself it also requires some co-ordination and management.

Zookeeper [45] is currently used as a distributed coordinator and Topic consumer offset manager by Kafka. Zookeeper keeps a parallel distributed synchronised file structure for parallel low latency data access, with single write operations directed to a 'leader node', responsible for forcing data propagation down to the available nodes for distribution. Future versions of Kafka are planned to phase out the use of Zookeeper and just utilise internal Topics for distributed coordination.

4.5. Evaluation of the parallel MC-NN implementation

4.5.1. Complexity of parallel MC-NN

The complexity definitions used in this section are given in Table 5. Extending the complexity analysis carried out in Section 3.2.1, our implementation scales with the available parallel hardware. As stated earlier, the number of Micro-Clusters remains fixed at the Maximum number allowed. Creation and deletion of Micro-Clusters is controlled by counters and pointers to the available pool of computational resources at run-time.

MC-NN Memory Size is $O(mx * c * d * p)$, or $O(mx * c * d)$ for each p Mapper. Additional processing time occurs due to more internal messaging and vote counting. For complexity of testing and training, each Micro-Cluster is calculated and the nearest Micro-Clusters' class label used $O((m * c * d) + (p * l))$ in parallel on p Mappers. In the best case, where there is only one Micro-Cluster per class label. The complexity will be $\Omega(ml + (p * l))$ or $\Omega(c + (p * l))$, each p Mapper running in parallel and the combined cost of their network voting. In the worst case this will be limited to the number of Micro-Clusters allowed by the algorithm $\Theta(mx * c + (p * l))$ for each Mapper p, and the combined latency induced by l messages.

4.5.2. Experimental setup

The setup of the proposed MC-NN algorithm over the cluster was implemented in a traditional MapReduce setup, which is depicted in Fig. 11.

Each of the Mappers (Hadoop nodes in the cluster) was initialised with a Samza container connecting them to a specific partition (distributed sub-section) of a Kafka topic stream. Each node when idle, waits for a message to arrive. Upon arrival of a message the node performs the required action (training,

Table 5
Parallel complexity definitions.

n	The number of instances in the data stream (only limited by experimentation as infinite in real data stream)
d	The number of dimensions of the data (number of attributes in instance)
c	Then number of class labels to classify
mx	The maximum number of Micro-Clusters (default set to 25 per class label)
m	The number of Micro-Clusters required to cover a specific data pattern
ml	The minimum number of Micro-Clusters (one Micro-Cluster per class label)
p	The number of parallel Mappers utilised
l	The network latency of a message

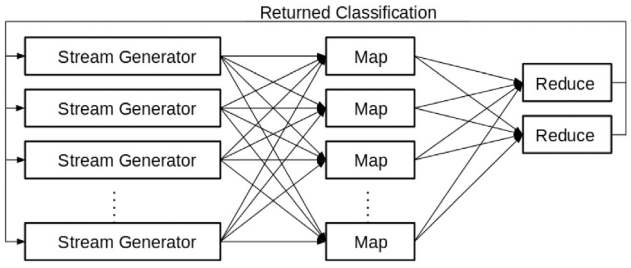


Fig. 11. Cluster setup.

predicting, etc.) of the message and data, then forwards the results to another Kafka Output stream; as depicted in Fig. 10. To stress test the scalability of the algorithm, multiple Stream Generators were instantiated, firing training and test data simultaneously to cluster. One of the nodes in the cluster was setup for Reducer jobs. The data that the Reducers receive and process is much less than that of the Mappers, as their task is a simple aggregation of local predictions from Mappers. Each Reducer was also setup with a Samza container connected to a Kafka stream.

The SEA data stream [39], Hyperplane data stream and HAR data set were used in multiple configurations of stream generators and different numbers of Mappers. A brief description of the generators can be found in Section 3.2.2. The cluster utilised had 16 physical worker nodes and 1 master node each with a quad core processor, giving a Hadoop virtual pool of 64 worker nodes. The parallelisation was limited to 1 node per machine to stop data bottlenecks at the network level. Two configurations of Stream Generators were created, 4 and 8 to send data in parallel to the Mappers. Each stream generator was pre-configured to launch one of the data streams (SEA, HYP or HAR). The total stream length was divided by the number of stream generators so that speed comparisons could be made upon a fixed stream length processed over different configurations. For each of the stream generator configurations 5 Mapper configurations were created and evaluated: 1,3,5,7 and 9. The number of Mappers here controls the level of parallelism within the system. The reason for using odd numbers of Mappers for the parallelism is so that voting should have an outright winning classification.

A 10 million data instance stream was created in parallel across the Stream Generators. To highlight the parallel performance of the system 'batching' was utilised to reduce the latency communication costs. Training messages created were batched into individual 10 thousand blocks for the Kafka messaging streams. The HAR data is looped instance by instance from each of the stream generators, to mimic the presence of a much larger (infinite) data stream for performance evaluation.

For each parallel Mapper configuration an equally parallel Kafka distributed Topic was created for the Samza containers to join to on a 1:1 basis. In addition to each Samza container joining to a specific Topic partition, a shared broadcast Topic was used for distributing the test instances to all Mappers at the same time as depicted in Fig. 9. The training and test messages were created simultaneously in parallel within their own local Stream Generators, with 10 thousand training instances to 1 test message produced.

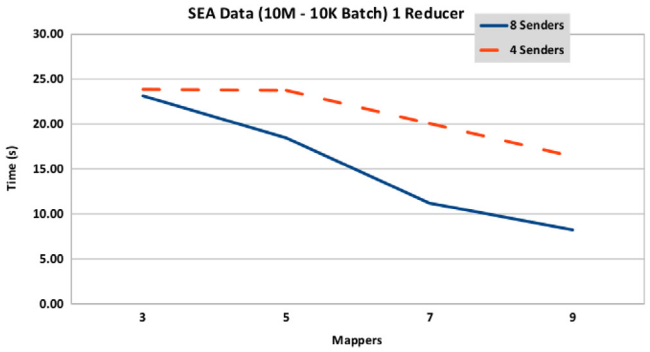


Fig. 12. Scalability of parallel MC-NN tested on the SEA data stream (10 million instances).

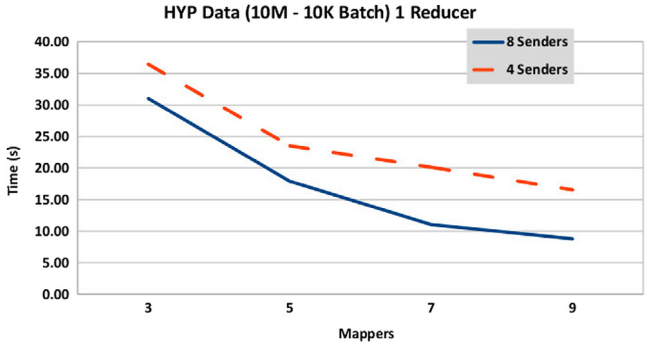


Fig. 13. Scalability of parallel MC-NN tested on the Hyperplane data stream (10 million instances).

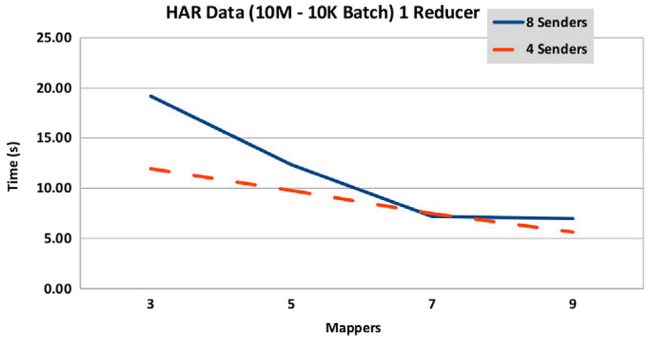


Fig. 14. Scalability of parallel MC-NN tested on the HAR data stream (10 million instances).

4.5.3. Scalability results for the parallel MC-NN implementation

Fig. 12 shows the overall reduction in processing time with both more Stream Generators and the use of more parallel processors. Note that experiments with only 1 Mapper have been attempted, however, a data stream with 10 million data instances sent in parallel to a single node caused the Samza containers to re-initialise — forced by the management software.

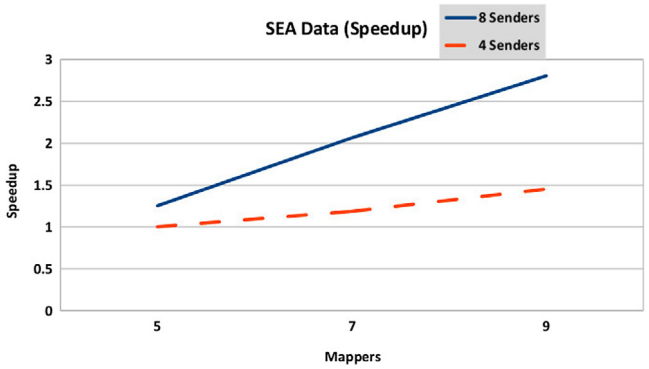


Fig. 15. Speedup of SEA data set with increasing Mappers.

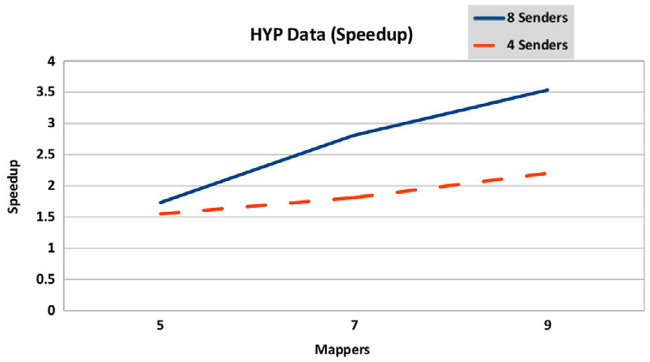


Fig. 16. Speedup of HYP data set with increasing Mappers.

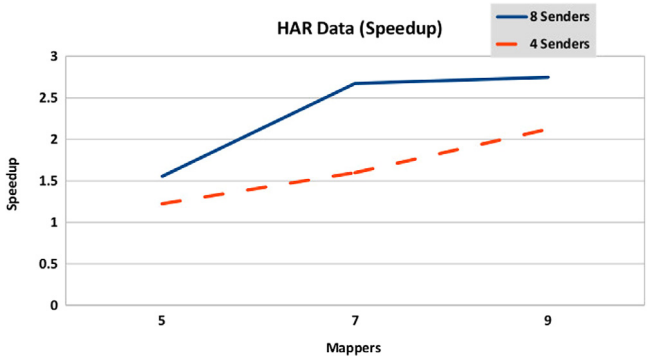


Fig. 17. Speedup of HAR data set with increasing Mappers.

It can be seen in Fig. 12, that MC-NN scales with respect to the number of MC-NN processing nodes utilised and the number of Stream Generators. The results for 3–5 Mappers with 4 stream generators shows little reduction in runtime. In the setup used in the SEA experiments, 4 generators running in parallel sending their data to 3–5 Mappers seems to be the physical limit encountered in terms of network performance. This is because the use of 4 generators can saturate the network, limiting the size of data that can be forwarded to the Mappers.

In Fig. 13 the time taken for processing 10 million instances reduces as both the number of Mappers and the stream generators increase. In the previous MC-NN experiments (Section 3.2.3), the Hyperplane generator stream can be efficiently classified with MC-NN with only a few Micro-Clusters required. The results presented in Fig. 13 show the same pattern. By utilising the minimum number of Micro-Clusters possible to 'fit' a data pattern, the testing time of an instance should be unrelated to the number of Mappers (each taking the same time in parallel). As the data is skewed in favour of one test instance to 10,000 training instances, the benefit of using parallel MC-NN can be seen. For example, as Mapper 1 receives one instance for training, Mapper 2 is free to receive the next training instance and performs its 'Training' process at the same time.

In Fig. 14 the time taken to process the data stream appears to be 'non-uniform' for using 7 Mappers. This can be explained by the fact that the HAR data set has been adapted to be a data stream. The actual patterns encoded in the data that lead to the class labels are not known. It is to be expected that when attempting to classify such a data set, it would be more or less efficient when using different parallel configurations, as the data will be distributed differently using varying configurations. For example, instances sent to the same Mappers will be incorporated locally into the Micro-Clusters with little to no processing. Differing instances will cause more Micro-Cluster splitting and take more time to train. Unfortunately, this is a case of there being no 'Magic Bullet' and certain data sets/streams will perform faster on different parallel configurations.

Overall, the pattern we can see is that the more MC-NN nodes that are used, the faster the processing of the 10 million data instances from the streams. What can also be seen is that a larger number of Stream Generators is also beneficial, which can be explained by a more distributed data communication load between the MC-NN nodes.

The speedup of the algorithm has been calculated by the following formula: $Speedup = \frac{T3}{TN}$ [46]. Where $T3$ is the baseline time taken for 3 Mappers and TN is the time taken for N Mappers. Typically, speedup is taken as the ratio of parallel processing time against a 'single' baseline, but as explained in Section 4.6 this was not possible. Fig. 15 shows the speedup of the algorithm on the SEA data set utilising multiple Mappers to spread the computational cost. With 4 data senders a speedup of approximately 1.5 can be seen, with 8 senders the speedup approaches 3. It should be re-iterated that these speedups are in comparison to 3 Mappers. Therefore a speedup factor of approximately 2.7 for 9 Mappers, means that 9 Mappers performed 2.7 times faster than 3 Mappers with 8 senders.

Table 6
Summarising percentage runtime improvement with respect to the number of stream generators and Mappers.

Number of stream generators	Number of Mappers	SEA %	HYP %	HAR %
4	5	0.4	55	22
4	7	19	81	60
4	9	45	120	112
8	5	25	73	55
8	7	106	181	167
8	9	181	354	175

Fig. 16 shows the speedup of the algorithm on the HYP data set. For both 4 and 8 Senders configurations we can see an increase of the speedup in relation to the number of Mappers used.

Fig. 17 shows the speedup of the algorithm on the HAR data set. An overall increase can be observed as the number of Mappers is increased. A significant gradient reduction can be seen with 7–9 Mappers with 8 senders. For the HAR data set to be used as a data stream within the stream generators it was pre-loaded into memory before the data stream 'ran', otherwise there would have been costly IO operations from the hard disk. As each data instance was held in memory creating an instance was extremely fast. The gradient reduction that can be seen in the figure can be explained by the fact that the physical limit of the network configuration was reached. This is because, the 8 senders to 7–9 Mappers is approaching at a 1:1 ratio.

For better readability the speedup shown in Figs. 15–17 is also shown in Table 6 as percentage improvement of runtime based on 3 Mappers.

4.6. Implementation, configuration and experimental issues

The current implementation has been *realised* by combining a new classifier implementation with multiple Open Source technologies. While Open Source technologies (namely, Hadoop, Samza and Kafka) have considerable advantages for reducing development time frames (and ensuring that the outcome can be shared with a large user community), their use and practical deployment is rarely complication free. From Figs. 12–14 it should be noted that there is no 1 Mapper configuration. During experimentation the authors noticed a repeated node failure, leading to a re-submission for a sustained data stream to flow through this specific configuration. The Samza container managing the 1 Mapper repeatedly failed within the Hadoop framework. This occurred due to Samza attempting to buffer the entire data stream in memory on a single node, leading to possible unpredictable errors with variable sized data streams and available memory on the nodes of the computational cluster. Distributed/parallel technologies are well documented and aware of external and partial failures over clustered machines and partitioned/'sharded' data. Most technologies utilise some strategy to deal with failures and data recovery. RAID disks, Hadoop's HDFS and Kafka use data replication, holding data in multiple locations simultaneously. Data processing failures such as Hadoop node failures are handled by the Node Manager. As Hadoop was primarily designed to process batch data in individual 'embarrassingly parallel' blocks, a failed task can simply be re-scheduled to be performed on another node. Similarly, Samza '*masquerades*' as a Hadoop job of unbounded length, only processing data when it becomes available. When a Samza node fails it is re-started by Hadoop as another job. While batch processing can be processed in disjointed iterative sub-tasks, data stream mining requires that the data streams are processed in the order and speed that they are made available to the classifier. This is especially true when the data is generated from different sources in parallel. By restarting a Samza job within the Hadoop framework a decision must be made as to how data stream processing will proceed. The original Kafka data stream does not know that the processing container failed, it simply keeps a running pointer of the last message in the linear stream that it gave to the processing job. Here only 2 choices are available:

(1) Re-read the entire data stream (limited to Kafka logs).
(2) Start a completely new job from this point and accept that the previous data was lost.

Re-reading the entire data stream will depend on the pre-set Kafka configurations and how long the data stream has been running for (perhaps, weeks or months). Kafka only retains messages for a pre-set period of time or until messages are overwritten on a 'rolling log' format of fixed hard disk size, whichever happens sooner. Starting a new job from the point of failure makes the classification analysis with other configurations incomparable. Only a subset of the data stream is processed by the new job. Accuracies and processing times are no longer comparable. For the purpose of this paper the Kafka nodes only kept messages for a maximum of 1 h. This allowed any old data to be naturally flushed out of the system in a reasonable time, for other experiments to be executed.

5. Conclusions

The development of a new parallel adaptive data stream classifier for data streams, termed MC-NN, is presented. This research is motivated by the fact that very little work has been conducted on the development of real-time scalable parallel data stream classification, even though many applications with high throughput data streams exist. MC-NN is naturally parallel and has been implemented on a computer cluster.

MC-NN realises real-time adaptation of data stream statistics using a novel implementation of Micro-Cluster and a Nearest Neighbour classification approach. Loosely speaking, Micro-Clusters adapt quickly to concept drifts through splitting into new Micro-Clusters based on variance and misclassification errors. Adaptation and learning within the cluster is performed through the deletion of Micro-Clusters with a low participation. A Micro-Cluster is considered participating if it regularly absorbs new data instances by being correctly situated over a data pattern. An empirical evaluation of MC-NN showed that it is competitive in terms of classification accuracy and adaptation to concept drifts with other existing popular data stream classifiers, i.e. Hoeffding Trees, adaptive Naïve Bayes and real-time KNN. The results show that MC-NN has a similar or better overall classification accuracy compared with its competitors and better adaptability to concept drifts. Furthermore, the results show shorter runtime of MC-NN compared with its competitors.

Parallelisation of MC-NN is achieved by distributing Micro-Clusters to computational nodes in a computer cluster. Adaptation and training is achieved by concurrently distributing training instances from the data stream evenly among the computational nodes in the cluster. Each node then trains and adapts to concept drift in the same way as the serial implementation of MC-NN. Classification is achieved through the use of a voting mechanism by each computational node. An architecture that allows the parallel processing of data streams has been realised and implemented. The architecture is based on integrating various distributed Open Source technologies such as Hadoop, Samza and Kafka. The paper describes the use of these technologies for parallel data stream processing and highlights issues and experiences. An empirical evaluation of the parallel MC-NN implementation utilising multiple data streams of varying attribute and class label sizes, shows that parallel MC-NN scales well with respect to the number of computational nodes utilised and the amount of Data Stream Generators used.

Acknowledgements

This research has been supported by the UK Engineering and Physical Sciences Research Council (EPSRC) grant EP/L505043/1.

References

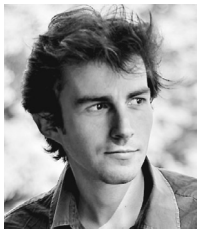
[1] M. Ebbers, A. Abdel-Gayed, V. Budhi, F. Dolot, Addressing Data Volume, Velocity, and Variety with IBM InfoSphere Streams V3.0, 2013.
[2] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: PODS, 2002, pp. 1–16.
[3] M.M. Gaber, A. Zaslavsky, S. Krishnaswamy, Mining data streams: a review, ACM SIGMOD Rec. 34 (2005) 18–26.
[4] M. Gaber, A. Zaslavsky, S. Krishnaswamy, A survey of classification methods in data streams, Data Streams (2007) 39–59.
[5] J. Gama, Knowledge Discovery from Data Streams, Chapman and Hall / CRC, 2010.
[6] T. Bujlow, T. Riaz, J. Pedersen, A method for classification of network traffic based on c5.0 machine learning algorithm, in: 2012 International Conference on Computing, Networking and Communications, ICNC, 2012, pp. 237–241.
[7] A. Jadhav, A. Jadhav, P. Jadhav, P. Kulkarni, A novel approach for the design of network intrusion detection system (NIDS), in: 2013 International Conference on Sensor Network Security Technology and Privacy Communication System, SNS PCS, 2013, pp. 22–27.
[8] A. Salazar, G. Safont, A. Soriano, L. Vergara, Automatic credit card fraud detection based on non-linear signal processing, in: 2012 IEEE International Carnahan Conference on Security Technology, ICCST, 2012, pp. 207–212.
[9] P. Domingos, G. Hulten, Mining high-speed data streams, in: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'00, ACM, New York, NY, USA, 2000, pp. 71–80.
[10] T. Le, F. Stahl, J.B. Gomes, M.M. Gaber, G.D. Fatta, Computationally efficient rule-based classification for continuous streaming data, in: Research and Development in Intelligent Systems XXXI, Springer International Publishing, 2014, pp. 21–34.

[11] J.a. Gama, P. Kosina, Learning decision rules from data streams, in: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two, IJCAI'11, AAAI Press, 2011, pp. 1255–1260.
[12] Y. Ben-Haim, E. Tom-Tov, A streaming parallel decision tree algorithm, J. Mach. Learn. Res. 11 (2010) 849–872.
[13] G.D.F. Morales, A. Bifet, Samoa: Scalable advanced massive online analysis, J. Mach. Learn. Res. 16 (2015) 149–153.
[14] A. Andrzejak, J. Gomes, Parallel concept drift detection with online map-reduce, in: 2012 IEEE 12th International Conference on Data Mining Workshops, ICDMW, 2012, 402–407. http://dx.doi.org/10.1109/ICDMW.2012.102.
[15] M. Tennant, F. Stahl, J. Gomes, Fast adaptive real-time classification for data streams with concept drift, in: Internet and Distributed Computing Systems, in: Lecture Notes in Computer Science, vol. 9258, Springer International Publishing, 2015, pp. 265–272.
[16] J. Gama, P. Medas, G. Castillo, P. Rodrigues, Learning with drift detection, in: Advances in Artificial Intelligence–SBIA 2004, Springer Berlin, Heidelberg, 2004, pp. 286–295.
[17] G.J. Ross, N.M. Adams, D.K. Tasoulis, D.J. Hand, Exponentially weighted moving average charts for detecting concept drift, Pattern Recognit. Lett. 33 (2) (2012) 191–198. http://dx.doi.org/10.1016/j.patrec.2011.08.019.
[18] R. Cavalcante, A. Oliveira, An approach to handle concept drift in financial time series based on extreme learning machines and explicit drift detection, in: 2015 International Joint Conference on Neural Networks, IJCNN, 2015, pp. 1–8. http://dx.doi.org/10.1109/IJCNN.2015.7280721.
[19] R.J. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, 1993.
[20] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995) 273–297. http://dx.doi.org/10.1023/A:1022627411411.
[21] M.A. Bramer, Automatic induction of classification rules from examples using N-Prism, in: Research and Development in Intelligent Systems XVI, Springer-Verlag, Cambridge, 2000, pp. 99–121.
[22] J. Han, M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann, 2001.
[23] P. Domingos, G. Hulten, A general framework for mining massive data streams, J. Comput. Graph. Stat. 12 (2008).
[24] C. Aggarwal, J. Han, J. Wang, P. Yu, A framework for clustering evolving data streams, in: Proceedings of the 29th VLDB Conference, Berlin Germany, 2003.
[25] P. Domingos, G. Hulten, Mining high-speed data streams, in: KDD, 2000, pp. 71–80.
[26] C.C. Aggarwal, J. Han, J. Wang, P.S. Yu, A framework for on-demand classification of evolving data streams, IEEE Trans. Knowl. Data Eng. 18 (5) (2006) 577–589.
[27] Esper (http://www.espertech.com/esper/seen November 2015).
[28] Samza (https://samza.apache.org/seen November 2015).
[29] R. Ranjan, Streaming big data processing in datacenter clouds, IEEE Cloud Comput. 1 (1) (2014) 78–83. http://dx.doi.org/10.1109/MCC.2014.22.
[30] M. Turilli, M. Santcroos, S. Jha, A comprehensive perspective on the pilot-job abstraction, CoRR abs/1508.04180. URL http://arxiv.org/abs/1508.04180.
[31] Rabbitmq (https://www.rabbitmq.com/seen November 2015).
[32] Kafka (http://kafka.apache.org/seen November 2015).
[33] Kestrel (https://twitter.github.io/kestrel/seen November 2015).
[34] Flume (https://flume.apache.org/seen November 2015).
[35] Scribe (https://github.com/facebookarchive/scribe/wikiseen November 2015).
[36] M. Tennant, F. Stahl, G. Di Fatta, J.B. Gomes, Towards a parallel computationally efficient approach to scaling up data stream classification, in: M. Bramer, M. Petridis (Eds.), Research and Development in Intelligent Systems XXXI, Springer International Publishing, 2014, pp. 51–65. http://dx.doi.org/10.1007/978-3-319-12069-0_4.
[37] M. Datar, A. Gionis, P. Indyk, R. Motwani, Maintaining stream statistics over sliding windows, in: ACM-SIAM Symposium on Discrete Algorithms, SODA 2002, 2002.
[38] Massive online analysis(http://moa.cms.waikato.ac.nz) (September 2016). URL http://moa.cms.waikato.ac.nz/.
[39] W. Street, Y. Kim, A streaming ensemble algorithm (SEA) for large-scale classification, in: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2001, pp. 377–382.
[40] A. Stisen, H. Blunck, S. Bhattacharya, T.S. Prentow, M.B. Kjærgaard, A. Dey, T. Sonne, M.M. Jensen, Smart devices are different: Assessing and mitigatingmobile sensing heterogeneities for activity recognition, in: Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems, SenSys'15, ACM, New York, NY, USA, 2015, pp. 127–140. URL http://doi.acm.org/10.1145/2809695.2809718.
[41] A. Dawid, Stastical Theory the Prequential Approach, vol. 147, The Royal Stastical Society, 1984, pp. 278–292.
[42] P. Sidhu, M. Bhatia, Empirical support for concept drifting approaches: Results based on new performance metrics, Int. J. Intell. Syst. Technol. Appl. 7 (6) (2015) 1–20.
[43] Kaggle competition winner 2014 (https://www.kaggle.com/c/lshtcseen November 2015).
[44] M. Diego, A. Bifet, M. Gianmarco, De Francisci, Random forests of very fast decision trees on gpu for mining evolving big data streams, in: Frontiers in Artificial Intelligence and Applications (ECAI), vol. 14, 2014, pp. 615–620.
[45] Zookeeper (http://zookeeper.apache.org/seen November 2015).
[46] M.A. Bramer, Principles of Data Mining, second ed., in: Undergraduate Topics in Computer Science, Springer, 2013.

Mark Tennant is currently working as Ph.D. student at the University of Reading in High Velocity Data Mining on Distributed Architectures. He received his degree in Artificial Intelligence and Cybernetics from the University of Reading in 2012. While working in Industry as a programmer, he became qualified as a Microsoft Professional (MP) and a Microsoft Certified Solutions Developer (MCSD).

Frederic Stahl is a Lecturer in Computer Science at the University of Reading. He holds a Ph.D. in Computer Science with the title "Parallel Rule Induction", which was awarded from the University of Portsmouth in 2010. His research interests include parallel data mining, data stream mining and data mining on smart devices.

Omer Rana is Professor of Performance Engineering in the School of Computer Science and Informatics at Cardiff University. He holds a Ph.D. in Neural and Parallel Systems from Imperial College, London. His research interests include high performance distributed systems, data mining/analysis and multi-agent systems.



João Bártolo Gomes is a principal investigator and lab head (distributed analytics) in the Data Analytics Department (DAD) with the Institute for Infocomm Research (I2R) under the Agency for Science, Technology and Research (A*Star), Singapore. Before, he was a member of the research group DAME (data mining engineering) at Universidad Politecnica de Madrid (UPM). His current research interests include ubiquitous knowledge discovery, machine learning algorithms, data stream mining and learning from evolving data streams.